A New Shape Subroutine for the Apple

by Richard T. Simoni Jr.

Athletes pole-vault, race cars spin, and fighter planes fire at enemy aircraft. Is this the real world? No, I'm talking about fast, smooth animation on the Apple II high-resolution graphics screen. In the past year, dozens of new Apple II programs have achieved such awesome animation capabilities that several years ago most Apple programmers would scarcely have believed them possible. After trying unsuccessfully to match the quality of the commercially produced animation in my own assembly-language programs, I realized that the problem stemmed from the standard Apple shape subroutine that I was using to display the shapes I wanted to animate.

Standard Hi-Res Package

The hi-res (high-resolution) graphics package I was using is the standard package supplied by Apple Computer. It once was supplied with all Apple II computers sold, and it can now be found on the volume 3 disk of the Apple Software Bank Contributed Programs, available from Apple dealers. Indeed, this package was eventually incorporated into the Applesoft language to add hi-res commands. Written in machine language, the package includes subroutines to clear the screen, plot a point, draw a line, and draw a shape on the hi-res screen. Although the clear, plot, and line subroutines work well in animation, the shape subroutine is much too slow to allow shapes to move across the screen quickly, smoothly, and without flickering.

The speed of the shape subroutine is the most important factor in animation for two main reasons. First, the speed with which the subroutine can plot the shape, erase it, and plot it again in its next position limits how fast any shape can move across the screen. Second, in a typical animation scheme, a shape moves from one position to the next in four phases, which correspond to the time required to plot the shape, the time the shape remains on the screen, the time required to erase the shape, and the time that the shape is not on the screen at all. These four phases repeat each time a shape moves to a new position. The time spent during each phase of the process determines how fast the shape moves and how smooth and flicker-free the animation looks. To maximize the smoothness, the time used in plotting the shape, erasing the shape, and leaving the shape off the screen must be minimized, for the human eye perceives these phases as contributing to the flicker of the image. On the other hand, if the amount of time the eye sees the image whole on the screen is significantly greater than the time required for the other phases, the image appears to move smoothly across the screen. Minimizing the time the image is totally off the screen is not difficult, for all calculations for the next plot can be done while the image is on the screen; when the image is erased, it can then be immediately plotted in the new position. The times required to plot and erase the shape, however, are directly determined by the speed of the image subroutine. If the subroutine is slow, the plot and erase times are long, and the image appears to flicker as it moves across the screen.

Representing Shapes

To understand why the standard Apple shape subroutine is too slow for most animation purposes, you must know how the subroutine works and especially how it expects a shape to be represented in memory. A shape is represented by a series of vectors in memory, with each vector specifying if a given pixel should be turned on. It also specifies which of the four adjacent pixels should be addressed by the next vector. This scheme best suits the representation of simple, single-line shapes such as those in figure 1. Unfortunately, if a shape must be filled in or if the shape has any detail drawn within its boundaries, as in figure 2, the shape's representation is awkward and inefficient at best. In these cases it is often necessary to overplot points and use many vectors that specify motion without plotting. Moreover, if the shape is large, the sheer size of

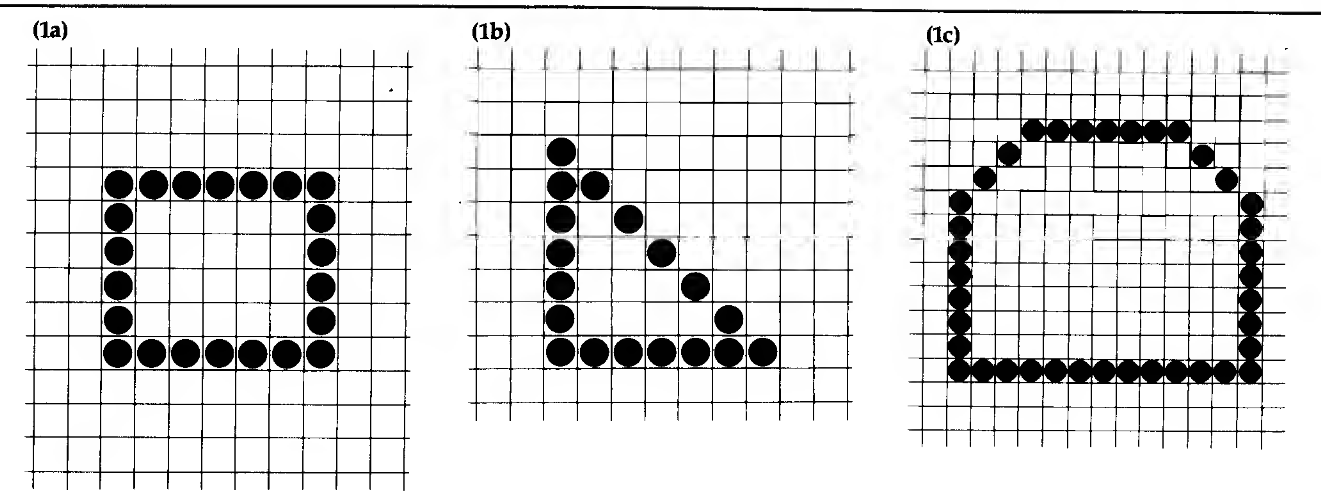


Figure 1: *Because they are easily represented in memory by a series of vectors, these simple single-line closed shapes are suitable for display by the standard Apple shape subroutine on the hi-res graphics screen.*

the vector table becomes unwieldy. When the time comes to plot these shapes, the subroutine steps through the table, and each vector takes up a certain amount of time. If the vector table represents the shape inefficiently, the end result is wasted time in the plotting of the shape.

Similarly contributing to the slow speed of the shape subroutine is the inclusion of scaling and rotation factors. In order to plot a shape, a calling routine must specify a scaling factor that determines the plotted shape's size (actual size, double size, triple size, etc.) and a rotation factor that determines the angle through which the shape is rotated before plotting. Although these factors are useful in some applications, using them with shape animation rarely produces satisfying results, and these calculations slow the subroutine considerably.

A New Shape Subroutine

After realizing that the speed bottleneck in my programs was caused by the shape subroutine, I went about designing my own subroutine with two criteria in mind. First, the subroutine had to be high speed to minimize image flicker, and second, the method of representing a shape in memory had to allow complicated images to be plotted as quickly as simple single-line shapes of the same overall size. One way to meet these criteria is to use a bit picture to represent the shape in memory. In other words, the shape is represented in main memory in the same form in which it is ultimately represented in the hi-res screen memory when the shape is plotted on the hi-res screen. Plotting the shape is then simple and fast: the bytes representing the shape in main memory need only be transferred to the hi-res screen memory. I used this technique in writing a fast shape subroutine suitable for animation.

The table of bytes that make up the bit picture is called the shape table.

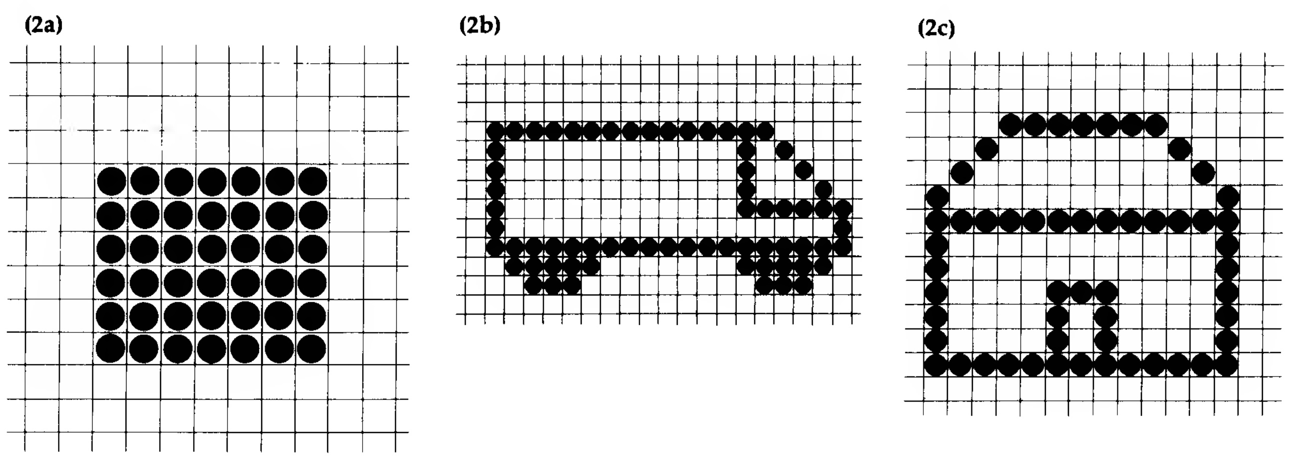
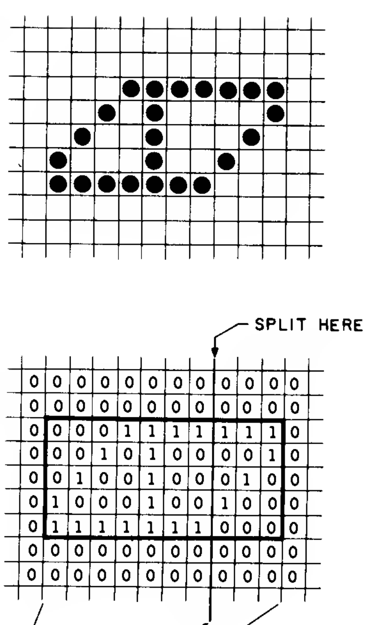


Figure 2: *The detail within these shapes makes their representation as vectors in memory inefficient; therefore, the standard Apple shape subroutine is neither well suited nor easy to use for the display of these shapes on the hi-res screen.*

(3a) (3b)




```
0001111   1110000
0010100   0010000
0100100   0100000
1000100   1000000
1111111   0000000
```

ADDED ZEROS

(3c)

```
1111000   0000111
0010100   0000100
0010010   0000010
0010001   0000001
1111111   0000000
```

(3d)

```
78   07
14   04
12   02
11   01
7F   00
```

Figure 3: *To form a shape table, start by drawing the desired shape on graph paper, using 1s and 0s to represent "on" and "off" pixels (3a). Next, split each line of bits into 7-bit groups, padding the last group of each line with 0s if necessary (3b). Then, reverse the order of the binary digits in each 7-bit group (3c) and convert to hexadecimal (3d). Later you must add height and width bytes as described in the text.*

A shape table is most easily formed through the use of the shape-editor program presented later in this article. To form a shape table manually, start by drawing the shape on a piece of graph paper with one pixel per square, as in figure 3a. Use 1s to represent on pixels and 0s to represent off pixels. Draw the smallest possible rectangle that still encloses

Listing 1: *A fast shape subroutine that plots high-resolution shapes on the Apple II.*

```
0000:                      OBJ $1B00
1B00:                      ORG $1B00           ;ASSEMBLY LOCATION
1B00:         *********************************************************
1B00:         * SHAPE SUBROUTINE WRITTEN BY RICHARD T. SIMONI, JR.    *
1B00:         *                                                       *
1B00:         * SHAPE WORKS BY STEPPING THROUGH THE USER TABLE ONE    *
1B00:         * HI-RES LINE AT A TIME, SHIFTING THE BIT PATTERN THE   *
1B00:         * APPROPRIATE NUMBER OF TIMES (DEPENDING ON THE         *
1B00:         * X-COORDINATE PASSED IN THE X- AND Y-REGISTERS), AND   *
1B00:         * MOVING THE PATTERN TO THE PROPER PLACE IN THE HI-RES  *
1B00:         * SCREEN MEMORY.                                        *
1B00:         *********************************************************
1B00:         STARTZ     EQU $19             ;START OF LINE STORAGE
1B00:         YCOORD     EQU $E3             ;LINE COUNTER
1B00:         START      EQU $EB             ;USER TABLE POINTER
1B00:         ADDRL      EQU $ED             ;1ST SCREEN BYTE TO USE
1B00:         ADDRH      EQU $EE             ;  IN LINE YCOORD
1B00:         ADDRADD    EQU $EF             ;OFFSET FROM LEFT BYTE
1B00:         SHFTNUM    EQU $F9             ;NUMBER OF SHIFTS
1B00:         ENDLN      EQU $FD             ;LAST LINE + 1
1B00:         WIDTH      EQU $FB             ;WIDTH OF USER TABLE
1B00:         INDEX      EQU $FC             ;POINTER IN USER TABLE
1B00:         *
1B00:         * DIVIDE X-COORD BY 7 TO GET BYTE OFFSET FROM LEFTMOST
1B00:         * BYTE IN ANY HI-RES LINE. REMAINDER WILL BE CORRECT
1B00:         * NUMBER OF SHIFTS TO PERFORM ON BIT PATTERN.
1B00:         * DIVISION IS PERFORMED USING LOOKUP TABLE FOR SPEED.
1B00:         *
1B00: 85 E3              STA YCOORD          ;STORE Y-COORD (COUNTER)
1B02: 8A                 TXA
1B03: 0A                 ASL A
1B04: AA                 TAX
1B05: 98                 TYA
1B06: 2A                 ROL A
1B07: A8                 TAY                 ;MULTIPLY X-COORD BY TWO
1B08: 18                 CLC
1B09: 8A                 TXA                 ;A-REG = X-COORD*2 LO-BYTE
1B0A: 69 83              ADC #>QUOTBL        ;ADD TABLE ADDRESS LO-BYTE
1B0C: 85 ED              STA ADDRL           ;STORE RESULT
1B0E: 98                 TYA                 ;A-REG = X-COORD*2 HI-BYTE
1B0F: 69 1B              ADC #<QUOTBL        ;ADD TABLE ADDRESS HI-BYTE
1B11: 85 EE              STA ADDRH           ;STORE RESULT
1B13: A0 00              LDY #$00            ;ZERO Y-REG FOR INDEXING
1B15: B1 ED              LDA (ADDRL),Y       ;LOAD X-COORD/7 FROM TABLE
1B17: 85 EF              STA ADDRADD         ;ADDRADD = X-COORD/7
1B19: C8                 INY                 ;REMAINDER FOLLOWS IN TABLE
1B1A: B1 ED              LDA (ADDRL),Y       ;LOAD REMAINDER FROM TABLE
1B1C: 85 F9              STA SHFTNUM         ;SHFTNUM = REMAINDER
1B1E:         *
1B1E:         * INITIALIZE LOCATIONS ENDLN AND WIDTH. ENDLN CONTAINS
1B1E:         * THE Y-COORD OF THE LAST LINE + 1. WIDTH CONTAINS THE
1B1E:         * WIDTH (IN BYTES) OF EACH LINE.
1B1E:         *
1B1E: A5 E3              LDA YCOORD
1B20: A0 00              LDY #$00
1B22: 18                 CLC
1B23: 71 EB              ADC (START),Y
1B25: 85 FD              STA ENDLN           ;ENDLN = Y-COORD+LENGTH
1B27: C8                 INY
1B28: B1 EB              LDA (START),Y
1B2A: 85 FB              STA WIDTH           ;GET & STORE WIDTH
1B2C: C8                 INY
1B2D: 84 FC              STY INDEX           ;INDEX=2
1B2F:         *
1B2F:         * LOOP1 IS THE LOOP THAT IS CYCLED THROUGH ONCE FOR EACH
1B2F:         * LINE ON THE HI-RES SCREEN
1B2F:         *
1B2F: A6 FB   LOOP1      LDX WIDTH           ;X-REG=0 (COUNTER)
1B31: A4 FC              LDY INDEX
1B33:         *
1B33:         * MOVE BYTES FOR LINE YCOORD FROM USER TABLE TO ZERO PAGE
1B33:         *
1B33: B1 EB   LOOP2      LDA (START),Y       ;GET XTH BYTE OF LINE
1B35: 95 19              STA STARTZ,X        ;STORE IN STARTZ+X
1B37: C8                 INY
1B38: CA                 DEX                 ;MOVED ALL BYTES YET?
1B39: D0 F8              BNE LOOP2           ;NO, LOOP
1B3B: 86 19              STX STARTZ          ;STARTZ=0
1B3D: 84 FC              STY INDEX
1B3F:         *
1B3F:         * SHIFT THE BIT PATTERN SHFTNUM TIMES
1B3F:         *
1B3F: A4 F9              LDY SHFTNUM         ;IS SHFTNUM=0?
1B41: F0 16              BEQ SKIP            ;YES, SKIP THE SHIFTING
1B43: 18      LOOP3      CLC                 ;NO, START SHIFTING
1B44: A6 FB              LDX WIDTH
1B46: 08                 PHP                 ;KEEP STACK IN ORDER
1B47: 28      LOOP4      PLP                 ;RESTORE CARRY
1B48: B5 19              LDA STARTZ,X        ;LOAD ORIGINAL PATTERN
1B4A: 2A                 ROL A
```

Listing 1 continued on page 300

the entire figure. Then split each line of binary digits enclosed by the rectangle into 7-bit groups. If, as in figure 3b, the last group doesn't have a full 7 bits, add enough 0s to the end of each line to bring the total up to 7 bits. Due to limitations to the subroutine, no more than seven 7-bit groups per line are allowed. Reverse the order of the bits in each group, as shown in figure 3c. Convert each new 7-bit group into its hexadecimal or decimal equivalent, whichever is preferred (figure 3d shows the hexadecimal equivalent) and, reading across each line left to right from the top to the bottom line, recopy the list of numbers in table (linear) form. The table is now complete except for two bytes that belong at the top of the table. The first of these bytes represents the height of the shape—in other words, the number of lines of digits in figure 3b; the second byte represents the width of the shape in 7-bit groups—that is, the number of 7-bit groups used in each line in figure 3b. As previously mentioned, this width should be no more than seven groups. The complete table in hexadecimal form for the sample shape used in figure 3 is as follows:

05 02 78 07 14 04 12 02
11 01 7F 00

The shape subroutine itself is shown in listing 1, and the lookup tables used by the subroutine are shown in listing 2. Before calling the subroutine, several registers and memory locations must be set up with certain parameters, including the hi-res screen coordinates of the pixel where the upper left corner of the bit picture should be positioned. The low-order byte of the x-coordinate should be placed in the X register, and the corresponding high-order byte of the x-coordinate (either 1 or 0) goes in the Y register. The y-coordinate goes in the A register (accumulator). The low- and high-order bytes of the shape-table starting address should be stored in hexadecimal memory locations EB and EC, respectively. The subroutine can then be called with the usual JSR instruc-

Text continued on page 303

Listing 1 continued:

```
1B4B: 2A                ROL A              ;ROTATF LEFT TWICF
1B4C: 08                RHR                ;SAVF CARRY
1B4D: 4A                LSR A              ;SHIFT RIGHT ONCF
1B4F: 95 19             STA STARTZ,X       ;STORE SHIFTED RATTERN
1B50: CA                DEX
1B51: F0 FF             CRX #$FF           ;ROTATFD EACH BYTE?
1B53: D0 F2             BNF LOOP4          ;NO, LOOP
1B55: 28                RLR                ;KFFR STACK IN ORDFR
1B56: 88                DFY
1B57: D0 EA             BNE LOOR3          ;LOOR IF Y<>0
1B59:               *
1B59:               * CALCULATF HI-RFS SCRFEN ADDRESS FOR FIRST BYTE TO
1B59:               * BE USED IN LINE YCOORD
1B59:               *
1B59: A4 E3         SKIR      LDY YCOORD
1B5B: B9 B3 1D                LDA LOSTRT,Y
1B5F: 18                      CLC
1B5F: 65 EF                   ADC ADDRADD
1B61: 85 FD                   STA ADDRL
1B63: B9 73 1E                LDA HISTRT,Y
1B66: 69 00                   ADC #$00
1B68: 85 FF                   STA ADDRH          ;GFT ADDR FOR 1ST BYTF
1B6A:               *
1B6A:               * MOVF SHIFTFD BYTFS FROM ZFRO PAGF TO HI-RFS SCRFFN
1B6A:               * MFMORY. FOR NON-FXCLUSIVE-OR RLOTTING, CHANGF
1B6A:               * 'FOR (ADDRL),Y' TO 'ORA (ADDRL),Y' (ORCODE $11).
1B6A:               *
1B6A: A0 00                   LDY #$00
1B6C: A6 FB                   LDX WIDTH
1B6E: B5 19         LOOR5     LDA STARTZ,X
1B70: 51 FD                   FOR (ADDRL),Y
1B72: 91 ED                   STA (ADDRL),Y      ;RLOT BYTE ON SCREFN
1B74: C8                      INY
1B75: CA                      DFX
1B76: F0 FF                   CRX #$FF           ;THROUGH RLOTTING LINE?
1B78: D0 F4                   BNE LOOR5          ;NO, LOOR
1B7A: F6 E3                   INC YCOORD         ;YES, GO TO NFXT LINE
1B7C: A5 F3                   LDA YCOORD
1B7F: C5 FD                   CMR ENDLN          ;MORE LINFS?
1B80: D0 AD                   BNE LOOR1          ;YFS, LOOR
1B82: 60                      RTS                ;NO, RETURN
1B83:               QUOTBL    FQU *
1B83:               LOSTRT    FQU *+560
1B83:               HISTRT    FQU *+752

*** SUCCFSSFUL ASSEMBLY: NO ERRORS
```

Listing 2: *Lookup tables used by the listing 1 subroutine.*

```
1B83- 00 00 00 01 00 02 00 03 00 04 00 05 00
1B90- 06 01 00 01 01 01 02 01 03 01 04 01 05 01 06 02
1BA0- 00 02 01 02 02 02 03 02 04 02 05 02 06 03 00 03
1BB0- 01 03 02 03 03 03 04 03 05 03 06 04 00 04 01 04
1BC0- 02 04 03 04 04 04 05 04 06 05 00 05 01 05 02 05
1BD0- 03 05 04 05 05 05 06 06 00 06 01 06 02 06 03 06
1BE0- 04 06 05 06 06 07 00 07 01 07 02 07 03 07 04 07
1BF0- 05 07 06 08 00 08 01 08 02 08 03 08 04 08 05 08
1C00- 06 09 00 09 01 09 02 09 03 09 04 09 05 09 06 0A
1C10- 00 0A 01 0A 02 0A 03 0A 04 0A 05 0A 06 0B 00 0B
1C20- 01 0B 02 0B 03 0B 04 0B 05 0B 06 0C 00 0C 01 0C
1C30- 02 0C 03 0C 04 0C 05 0C 06 0D 00 0D 01 0D 02 0D
1C40- 03 0D 04 0D 05 0D 06 0E 00 0E 01 0E 02 0E 03 0E
1C50- 04 0E 05 0E 06 0F 00 0F 01 0F 02 0F 03 0F 04 0F
1C60- 05 0F 06 10 00 10 01 10 02 10 03 10 04 10 05 10
1C70- 06 11 00 11 01 11 02 11 03 11 04 11 05 11 06 12
1C80- 00 12 01 12 02 12 03 12 04 12 05 12 06 13 00 13
1C90- 01 13 02 13 03 13 04 13 05 13 06 14 00 14 01 14
1CA0- 02 14 03 14 04 14 05 14 06 15 00 15 01 15 02 15
1CB0- 03 15 04 15 05 15 06 16 00 16 01 16 02 16 03 16
1CC0- 04 16 05 16 06 17 00 17 01 17 02 17 03 17 04 17
1CD0- 05 17 06 18 00 18 01 18 02 18 03 18 04 18 05 18
1CE0- 06 19 00 19 01 19 02 19 03 19 04 19 05 19 06 1A
1CF0- 00 1A 01 1A 02 1A 03 1A 04 1A 05 1A 06 1B 00 1B
1D00- 01 1B 02 1B 03 1B 04 1B 05 1B 06 1C 00 1C 01 1C
1D10- 02 1C 03 1C 04 1C 05 1C 06 1D 00 1D 01 1D 02 1D
1D20- 03 1D 04 1D 05 1D 06 1F 00 1F 01 1E 02 1E 03 1F
1D30- 04 1E 05 1F 06 1F 00 1F 01 1F 02 1F 03 1F 04 1F
1D40- 05 1F 06 20 00 20 01 20 02 20 03 20 04 20 05 20
1D50- 06 21 00 21 01 21 02 21 03 21 04 21 05 21 06 22
1D60- 00 22 01 22 02 22 03 22 04 22 05 22 06 23 00 23
1D70- 01 23 02 23 03 23 04 23 05 23 06 24 00 24 01 24
1D80- 02 24 03 24 04 24 05 24 06 25 00 25 01 25 02 25
1D90- 03 25 04 25 05 25 06 26 00 26 01 26 02 26 03 26
1DA0- 04 26 05 26 06 27 00 27 01 27 02 27 03 27 04 27
1DB0- 05 27 06 00 00 00 00 00 00 00 00 80 80 80 80 80
1DC0- 80 80 80 00 00 00 00 00 00 00 00 80 80 80 80 80
1DD0- 80 80 80 00 00 00 00 00 00 00 00 80 80 80 80 80
1DE0- 80 80 80 00 00 00 00 00 00 00 00 80 80 80 80 80
1DF0- 80 80 80 28 28 28 28 28 28 28 28 A8 A8 A8 A8 A8
1E00- A8 A8 A8 28 28 28 28 28 28 28 28 A8 A8 A8 A8 A8
1E10- A8 A8 A8 28 28 28 28 28 28 28 28 A8 A8 A8 A8 A8
1E20- A8 A8 A8 28 28 28 28 28 28 28 28 A8 A8 A8 A8 A8
1E30- A8 A8 A8 50 50 50 50 50 50 50 50 D0 D0 D0 D0 D0
1E40- D0 D0 D0 50 50 50 50 50 50 50 50 D0 D0 D0 D0 D0
```

Listing 2 continued on page 303

(1a)

Coordinate	6502 Register
x low-order byte	X
x high-order byte	Y
y	A

(1b)

Address Byte	Memory Location
low-order byte	EB
high-order byte	EC

Table 1: *Summary of parameters that must be set up prior to calling the shape subroutine: coordinates of upper left corner of bit picture (1a) and starting address (hexadecimal) of shape table (1b).*

Text continued from page 298:

tion. A summary of the parameter setup is given in table 1.

The subroutine works by taking the exclusive-OR of each affected bit in page-1 hi-res screen memory with the corresponding bit of the bit picture. This exclusive-OR plotting has several advantages. First, a color need not be specified; the shape is drawn by calling the subroutine once and is erased by simply calling it again with the same screen coordinates. Second, any shape drawn using exclusive-OR plotting is nondestructive; that is, whatever the shape happens to plot over is restored when the shape is erased. This property can be used to form interesting backgrounds that need not be redrawn after shapes are plotted and moved on top of them. Cross-hair cursors are also free to move around without destroying the screen's previous contents.

Several details about the subroutine need to be explained. Zero page (hexadecimal locations 00 through FF) of memory is used for temporary storage; the particular locations used were chosen to avoid destruction of locations used by the Apple Monitor, Applesoft, Integer Basic, and the DOS (disk operating system). The subroutine does not operate correctly without the tables shown in listing 2. These tables may be stored anywhere in memory, but are best located immediately after the subroutine in memory. Three pertinent

Text continued on page 306

Listing 2 continued:

```
1E50- D0 D0 D0 50 50 50 50 50 50 50 50 D0 D0 D0 D0 D0
1E60- D0 D0 D0 50 50 50 50 50 50 50 50 D0 D0 D0 D0 D0
1E70- D0 D0 D0 20 24 28 2C 30 34 38 3C 20 24 28 2C 30
1E80- 34 38 3C 21 25 29 2D 31 35 39 3D 21 25 29 2D 31
1E90- 35 39 3D 22 26 2A 2E 32 36 3A 3E 22 26 2A 2E 32
1EA0- 36 3A 3E 23 27 2B 2F 33 37 3B 3F 23 27 2B 2F 33
1EB0- 37 3B 3F 20 24 28 2C 30 34 38 3C 20 24 28 2C 30
1EC0- 34 38 3C 21 25 29 2D 31 35 39 3D 21 25 29 2D 31
1ED0- 35 39 3D 22 26 2A 2E 32 36 3A 3E 22 26 2A 2E 32
1EE0- 36 3A 3E 23 27 2B 2F 33 37 3B 3F 23 27 2B 2F 33
1EF0- 37 3B 3F 20 24 28 2C 30 34 38 3C 20 24 28 2C 30
1F00- 34 38 3C 21 25 29 2D 31 35 39 3D 21 25 29 2D 31
1F10- 35 39 3D 22 26 2A 2E 32 36 3A 3E 22 26 2A 2E 32
1F20- 36 3A 3E 23 27 2B 2F 33 37 3B 3F 23 27 2B 2F 33
1F30- 37 3B 3F
```

Listing 3: *This shape-editor program forms a shape table directly from a high-resolution screen image.*

```
100  TEXT : HOME : POKE  - 16298,0: POKE  - 16300,0
110  RESTORE : FOR I = 768 TO 774: READ J: POKE I,J: NEXT I: POKE 232,0: POKE 23
     3,3: DATA  1,0,3,0,45,5,0
120  DIM S%(105),T%(212)
130 XMAX = 42:YMAX = 35:ML = 101:MT = 10
140 H$ = "0123456789ABCDEF"
150 D$ =  CHR$ (4)
160  GOSUB 3100: GOSUB 3300: GOSUB 3400
400  REM  SHOW CURSOR POSITION DN GRID
410  XDRAW 1 AT CL + 1,CT + 3
420  REM  WAIT FOR KEYBOARD COMMAND
430 Q =  PEEK ( - 16384): IF Q < 128 THEN 430
440  POKE  - 16368,0:Q = Q - 128
500  REM
501  REM   CURSOR MOVEMENT COMMANDS
502  REM
510  IF Q <  >  ASC ("I") THEN 550
520  XDRAW 1 AT CL + 1,CT + 3
530  IF Y > 1 THEN Y = Y - 1:CT = CT - 4
540  GOTO 410
550  IF Q <  >  ASC ("M") THEN 590
560  XDRAW 1 AT CL + 1,CT + 3
570  IF Y < YMAX THEN Y = Y + 1:CT = CT + 4
580  GOTO 410
590  IF Q <  >  ASC ("J") THEN 630
600  XDRAW 1 AT CL + 1,CT + 3
610  IF X > 1 THEN X = X - 1:CL = CL - 4
620  GOTO 410
630  IF Q <  >  ASC ("K") THEN 700
640  XDRAW 1 AT CL + 1,CT + 3
650  IF X < XMAX THEN X = X + 1:CL = CL + 4
660  GOTO 410
700  REM
701  REM    PLOT COMMAND
702  REM
710  IF Q <  >  ASC ("P") THEN 810
720 ELE =  INT ((X - 1) / 14) + 3 * (Y - 1)
730 BIT = (X - 1) -  INT ((X - 1) / 14) * 14
740 A =  INT (S%(ELE) / 2 ^ BIT)
750  IF A / 2 <  >  INT (A / 2) THEN 810
760 S%(ELE) = S%(ELE) + 2 ^ BIT
770  FOR I = 2 TO 4: XDRAW 1 AT CL + 1,CT + I: NEXT I
780  HCOLOR= 7: HPLOT 29 + X,62 + Y
790  GOTO 430
800  REM
801  REM   ERASE COMMAND
802  REM
810  IF Q <  >  ASC ("E") THEN 900
820 ELE =  INT ((X - 1) / 14) + 3 * (Y - 1)
830 BIT = (X - 1) -  INT ((X - 1) / 14) * 14
840 A =  INT (S%(ELE) / 2 ^ BIT)
850  IF A / 2 =  INT (A / 2) THEN 900
860 S%(ELE) = S%(ELE) - 2 ^ BIT
870  FOR I = 2 TO 4: XDRAW 1 AT CL + 1,CT + I: NEXT I
880  HCOLOR= 0: HPLOT 29 + X,62 + Y
890  GOTO 430
900  REM
901  REM   CLEAR SCREEN COMMAND
902  REM
910  IF Q <  >  ASC ("C") THEN 1030
920  XDRAW 1 AT CL + 1,CT + 3
930  VTAB 23: PRINT "SURF YOU WANT TO FRASE THE SCRFEN?"
940  GOSUB 3500
950  VTAB 22: CALL  - 958: IF Q <  >  ASC ("Y") THFN 410
960  FOP I = 0 TO 105:S%(I) = 0: NFXT I
970  GOSUB 3300: GOSUB 3400: GOTO 410
1000  REM
1010  PEM   TABLE COMMAND
1020  PEM
1030  IF Q <  >  ASC ("T") THEN 1520
1040  VTAB 23: PPINT "SET CUPSOR TO TOP LEFT CORNER OF": PRINT "DFSIRED BIT MAP
     AND HIT PETURN";
1050 L5 = 1
```

Listing 3 continued on page 304

Listing 3 continued:

```
1060  GOTO 430
1070 PL = X:PT = Y
1080  VTAB 22: CALL  - 958: PRINT : PRINT "SET CURSOR TO BOTTOM RIGHT CORNER OF"
     : PRINT "DESIRED BIT MAP AND HIT RETURN";
1090 L5 = 2
1100  GOTO 430
1110 PR = X:PB = Y:L5 = 0
1120  XDRAW 1 AT CL + 1,CT + 3
1130  VTAB 22: CALL  - 958
1140  IF PL > PR OR PT > PB THEN  VTAB 23: HTAB 1: POKE 50,63: PRINT "ILLEGAL BI
     T MAP CORNERS": POKE 50,255: FOR I = 1 TO 2000: NEXT I: VTAB 22: CALL  - 95
     8: GOTO 410
1150  VTAB 23: HTAB 1: PRINT "NOW FORMING SHAPE TABLE"
1160  FOR I = 0 TO 212:T%(I) = 0: NEXT I
1170 L = PB - PT + 1:W = (PR - PL + 1) / 7: IF W <  >  INT (W) THEN W =  INT (W)
      + 1
1180 T%(0) = L:T%(1) = W:N = 2:Q = 0
1190  FOR Y = PT TO PB
1200  FOR X = PL TO PL + W * 7 - 1
1210  IF X > PR THEN BN = 0: GOTO 1250
1220 ELE =  INT ((X - 1) / 14) + 3 * (Y - 1)
1230 BIT = (X - 1) -  INT ((X - 1) / 14) * 14
1240 BN = 0:A =  INT (S%(ELE) / 2 ^ BIT): IF  INT (A / 2) <  > A / 2 THEN BN = 1
1250  IF BN = 1 THEN T%(N) = T%(N) + 2 ^ Q
1260 Q = Q + 1: IF Q = 7 THEN Q = 0:N = N + 1
1270  NEXT X: NEXT Y
1280  HOME : POKE  - 16303,0
1290  VTAB 2: PRINT "DO YOU WANT TO SEE THE TABLE IN HEX": PRINT "   OR IN DECIM
     AL?": PRINT : PRINT
1300  GOSUB 3500
1310  IF Q <  >  ASC ("D") AND Q <  >  ASC ("H") THEN 1280
1320 Z = 0: FOR I = 0 TO L * W + 1
1330 Z = Z + 1
1340  IF Q =  ASC ("D") THEN  PRINT  TAB( Z * 4);T%(I);: GOTO 1360
1350  PRINT  TAB( Z * 3); MID$ (H$, INT (T%(I) / 16) + 1,1); MID$ (H$,T%(I) -  I
     NT (T%(I) / 16) * 16 + 1,1);
1360  IF Z = 8 THEN Z = 0: PRINT
1370  NEXT I
1380  PRINT : PRINT : IF  PEEK (37) < 21 THEN  POKE 34, PEEK (37)
1390  PRINT "DO YOU WANT TO SAVE THE OBJECT TABLE": PRINT "   ON DISK?"
1400  GOSUB 3500
1410  IF Q <  >  ASC ("Y") THEN 1470
1420  PRINT : PRINT "WHAT DO YOU WANT TO NAME": INPUT "   THE FILE? ";N$
1430  FOR I = 0 TO L * W + 1: POKE 16384 + I,T%(I): NEXT I
1440  PRINT D$;"BSAVE";N$;",A$4000,L";L * W + 2
1450  PRINT "FILE SAVED USING NAME ";N$
1460  PRINT : PRINT : GOTO 1390
1470  POKE 34,0: HOME : VTAB 2: PRINT "DO YOU WANT TO RETURN TO THE": PRINT "
     SCREEN EDIT MODE?"
1480  GOSUB 3500
1490  IF Q <  >  ASC ("Y") THEN 2260
1500  GOSUB 3100: POKE  - 16304,0: GOSUB 3310: GOTO 410
1510  REM  'RETURN' PSEUDO-COMMAND
1520  IF Q <  > 13 THEN 1600
1530  ON L5 + 1 GOTO 430,1070,1110
1600  REM
1601  REM   SAVE TABLE COMMAND
1602  REM
1610  IF Q <  >  ASC ("S") THEN 1800
1620  XDRAW 1 AT CL + 1,CT + 3
1630  VTAB 23: INPUT "FILE NAME FOR SAVE? ";N$
1640  VTAB 24: PRINT "NOW SCANNING IMAGE";: HTAB 1
1650 Z1 = 0
1660  IF S%(Z1) = 0 AND Z1 < 105 THEN Z1 = Z1 + 1: GOTO 1660
1670 Z2 = 105
1680  IF S%(Z2) = 0 AND Z2 > 0 THEN Z2 = Z2 - 1: GOTO 1680
1690  IF Z1 > Z2 THEN Z1 = 0:Z2 = 1
1700  VTAB 24: PRINT "NOW SAVING IMAGE TO DISK";: VTAB 23: PRINT
1710  PRINT D$;"OPEN";N$: PRINT D$;"WRITE";N$
1720  PRINT Z1: PRINT Z2
1730  FOR I = Z1 TO Z2
1740  PRINT S%(I)
1750  NEXT I
1760  PRINT D$;"CLOSE";N$
1770  VTAB 22: CALL  - 958: GOTO 410
1800  REM
1801  REM   LOAD TABLE COMMAND
1802  REM
1810  IF Q <  >  ASC ("G") THEN 2100
1820  XDRAW 1 AT CL + 1,CT + 3
1830  VTAB 23: PRINT "SURE YOU WANT TO LOAD?"
1840  GOSUB 3500
1850  VTAB 22: CALL  - 958: IF Q <  >  ASC ("Y") THEN 410
1860  VTAB 23: INPUT "FILE NAME FOR LOAD? ";N$
1870  PRINT D$;"OPEN";N$: PRINT D$;"READ";N$
1880  INPUT Z1: INPUT Z2
1890  FOR I = 0 TO Z1:S%(I) = 0: NEXT I: FOR I = Z2 TO 105:S%(I) = 0: NEXT I
1900  FOR I = Z1 TO Z2
1910  INPUT S%(I)
1920  NEXT I
1930  PRINT D$;"CLOSE";N$
1940  GOSUB 3300: GOSUB 3400
1950  VTAB 22: CALL  - 958: VTAB 23: PRINT "NOW RETRACING IMAGE ON SCREEN"
1960 ELE = Z1:BIT = 0:CL = ML + 4 * ((ELE -  INT (ELE / 3) * 3) * 14)
1970 CT = MT + 4 *  INT (ELE / 3)
1980 A =  INT (S%(ELE) / 2 ^ BIT): IF  INT (A / 2) = A / 2 THEN 2000
1990  FOR I = 2 TO 4: XDRAW 1 AT CL + 1,CT + I: NEXT I: HPLOT 30 + (CL - ML) / 4
     ,63 + (CT - MT) / 4
```

Listing 3 continued on page 306

Listing 3 continued:

```
2000 CL = CL + 4:BIT = BIT + 1: IF BIT < > 14 THEN 1980
2010  IF ELE > = Z2 THEN  GOSUB 3310: GOTO 410
2020 BIT = 0:ELE = ELE + 1
2030  IF ELE / 3 =  INT (ELE / 3) THEN CL = ML:CT = CT + 4
2040  GOTO 1980
2100  REM
2101  REM   HELP COMMAND
2102  REM
2110  IF Q < > ASC ("H") AND Q < > ASC ("/") AND Q < > ASC ("?") THEN 2200
2120  VTAB 21: CALL  - 958: POKE  - 16303,0
2130  GOSUB 3170
2140  POKE  - 16304,0
2150  VTAB 20: PRINT : CALL  - 958: HTAB 2: PRINT "ACTUAL SIZE";: HTAB 21: PRINT
      "VIEWING WINDOW"
2160  GOTO 430
2200  REM
2201  REM   QUIT COMMAND
2202  REM
2210  IF Q < > ASC ("Q") THEN 430
2220  XDRAW 1 AT CL + 1,CT + 3
2230  VTAB 23: PRINT "SURE YOU WANT TO QUIT?"
2240  GOSUB 3500
2250  IF Q < > ASC ("Y") THEN VTAB 22: CALL  - 958: GOTO 410
2260  HOME : POKE  - 16303,0: POKE  - 16298,0: VTAB 24
2270  GOTO 9999
3000  REM
3010  REM   SUBROUTINES
3020  REM
3100  HOME
3110  HTAB 15: PRINT "COMMAND MENU": HTAB 15: PRINT "------- ----"
3120  VTAB 4: PRINT "I,J,K,M"; TAB( 9);"CURSOR MOVEMENT": PRINT : PRINT "P"; TAB
     ( 9);"PLOT POINT AT CURSOR POSITION": PRINT
3130  PRINT "E"; TAB( 9);"ERASE POINT AT CURSOR POSITION": PRINT : PRINT "C"; TA
     B( 9);"CLEAR SCREEN": PRINT
3140  PRINT "T"; TAB( 9);"MAKE SHAPE TABLE": PRINT : PRINT "S"; TAB( 9);"SAVE SH
     APE SOURCE FILE TO DISK": PRINT
3150  PRINT "G"; TAB( 9);"GET SHAPE SOURCE FILE FROM DISK": PRINT : PRINT "H OR
     ?"; TAB( 9);"HELP (RETURN TO THIS MENU)"
3160  PRINT : PRINT "Q"; TAB( 9);"QUIT PROGRAM EXECUTION"
3170  VTAB 24: HTAB 10: PRINT "HIT SPACE TO EXIT MENU";
3180  GOSUB 3500: IF Q < > ASC (" ") THEN 3180
3190  VTAB 21: CALL  - 958
3200  RETURN
3300  POKE 230,32: CALL 62450: HGR : SCALE= 1: ROT= 0
3310 PT = YMAX + 1:PB = 0:PL = XMAX + 1:PR = 0
3320  VTAB 21: HTAB 2: PRINT "ACTUAL SIZE";: HTAB 21: PRINT "VIEWING WINDOW";: C
     ALL  - 958: PRINT
3330 X =  INT (XMAX / 2):Y =  INT (YMAX / 2)
3340 MR = ML + XMAX * 4:MB = MT + YMAX * 4
3350 CL = ML + (X - 1) * 4:CT = MT + (Y - 1) * 4
3360  RETURN
3400  HCOLOR= 7
3410  FOR I = ML TO MR STEP 4: HPLOT I,MT TO I,MB: NEXT I
3420  FOR I = MT TO MB STEP 4: HPLOT ML,I TO MR,I: NEXT I
3430  RETURN
3500 Q =  PEEK ( - 16384): IF Q < 128 THEN 3500
3510  POKE  - 16368,0:Q = Q - 128
3520  RETURN
9999  END
```

Text continued from page 303:

tables are named QUOTBL, LOSTRT, and HISTRT. QUOTBL is a lookup table used internally by the subroutine to divide the *x*-coordinate by 7. LOSTRT and HISTRT are each 192 bytes long, and they contain the low- and high-order bytes of the address of the leftmost byte of each *y*-coordinate in page 1 of hi-res screen memory. For plotting on page 2 of the hi-res memory, a hexadecimal 20 should be added to each byte in the table HISTRT. Although I wanted the subroutine to be fully relocatable, I compromised this requirement in favor of additional speed. However, as I have written it, relocating the subroutine requires changing only the two locations referencing QUOTBL in lines 38 and 41 of listing 1.

A Note on Color

One of the most difficult aspects of using the Apple high-resolution graphics mode is trying to control the color of objects displayed on the screen. This difficulty arises because a color cannot be individually assigned to each pixel on the screen; the color depends instead on such factors as whether an object is drawn with pixels horizontally alternating between on and off and whether the on pixels have even or odd *x*-coordinates. Through careful programming and shape-table composition, you can control colors in this manner using the shape subroutine presented in this article. In newer Apples, however, two more colors are added to the hi-res screen by defining the previously unused high-order bit in each word in hi-res screen memory. Unfortunately, these colors cannot be easily displayed using the shape subroutine because the subroutine forces the extra bit in the hi-res screen to 0. For a complete description of color in the Apple hi-res screen, see page 19 of the *Apple II Reference Manual* (Cupertino: Apple Computer Inc., 1979).

The Shape-Editor Program

Although it is not difficult to form the shape table for a given shape, it is often time consuming. When writing a program that uses shapes, you rarely know in advance the exact pixel pattern that makes up the shape. Even if you know the pattern, you're probably not sure whether the shape will look good on the hi-res screen. It might take you hours to develop a suitable shape if you have to write out each trial on graph paper, form the shape table, and use the subroutine to display the shape before you can tell if it is satisfactory. This time-consuming method can bring the creative process to a halt. A more desirable situation would be one in which you could easily experiment with different shapes on the hi-res screen until you were satisfied with the results and then form the shape table directly from the screen image. I had this concept in mind when writing the shape-editor program shown in listing 3. The program features complete hi-res editing, both actual size and a blown-up view of the shape being drawn, disk storage of the current shape (the source file) for future editing, and assembly of a shape table from any portion of the current screen.

The editor program requires an Apple II with 32K bytes of memory, a disk drive, and Applesoft in ROM (read-only memory). When you run the program, the list of commands shown in photo 1 comes up on the screen. After you press the space bar, the left area of the screen becomes blank, and a grid appears on the right. The blank area is the "slate" on which you can draw different shapes actual size. Anything drawn also appears enlarged on the grid, making it easier to see details of the shape. Once the grid has been drawn, a

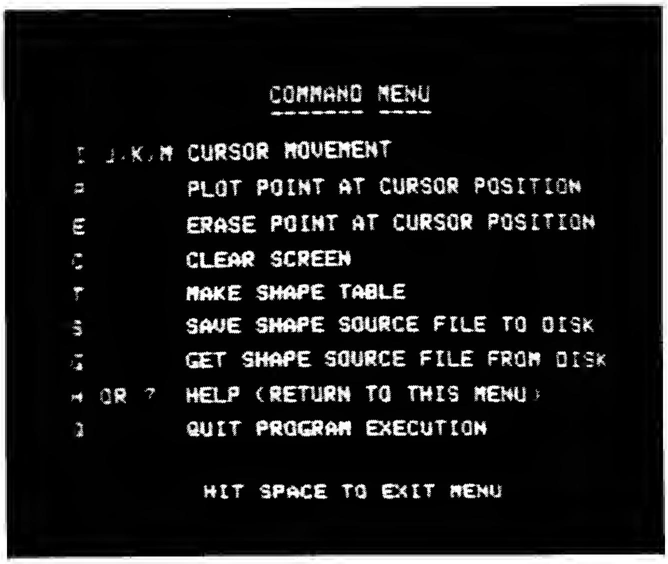


Photo 1: *The command menu that appears at the beginning of the shape-editor program (listing 3). This menu also appears whenever the Help key is pressed.*

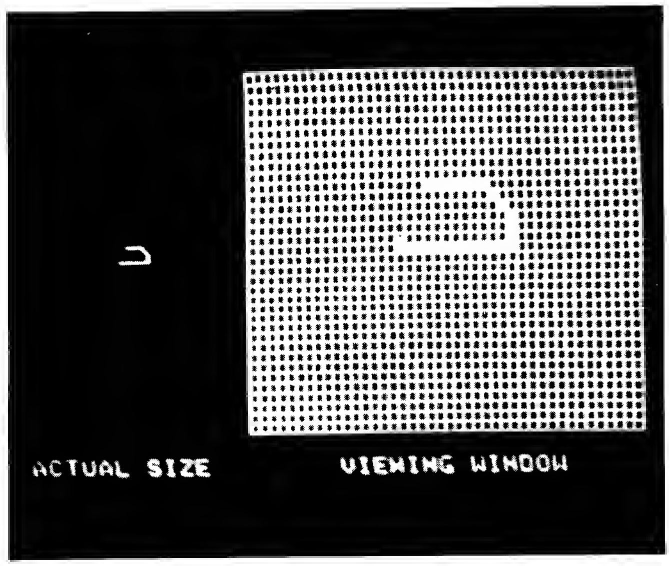


Photo 2: *A view of the screen-edit mode of the shape-editor program. The figure on the grid is an enlarged view of the actual-size shape on the left side of the screen. The cursor is the small horizontal line in a square above the lower left corner of the displayed shape.*

small horizontal line appears in one of the small squares in the grid. This is the cursor, which always shows the current drawing position of the program.

Once the cursor appears on the screen, you can execute any of the commands listed in the menu (photo 1) by pressing the corresponding letter on the keyboard. The letters I, J, K, and M are used for moving the graphics cursor up, left, right, and down, respectively. The Plot command plots a point at the current cursor position, and the Erase command erases the point at the current cursor position. Neither the Plot nor the Erase command causes any harm if the command has already been used at the cursor position (e.g., if the Plot command is used at a position where a point already exists). The Clear command clears the screen after prompting you to verify that the screen should indeed be cleared. By using the cursor-movement, Plot, Erase, and Clear commands, you can draw the desired shape on the screen and modify it as many times as necessary. A shape being drawn in this screen-edit mode is shown in photo 2.

With the Table command, you can make a shape table from any segment of the screen where you have drawn a shape. After choosing the Table command by pressing the T key, you must choose the smallest rectangle that encloses the shape; this is the same rectangle chosen when forming the shape table manually as previously described. You specify the boundaries of this rectangle by moving the cursor to the upper left position of the rectangle and pressing the Return key and then doing the same for the lower right corner of the rectangle. The corners are inclusive; that is, the rows and columns that contain the corners become the outermost edges included in the shape table. A portion of the rectangle selection process is shown in photo 3. After you select the desired rectangle, the program will form the shape table. The time this takes (typically 15 to 30 seconds) depends on the size of the shape. The completed shape table is displayed on the screen in either decimal or hexadecimal form, de-

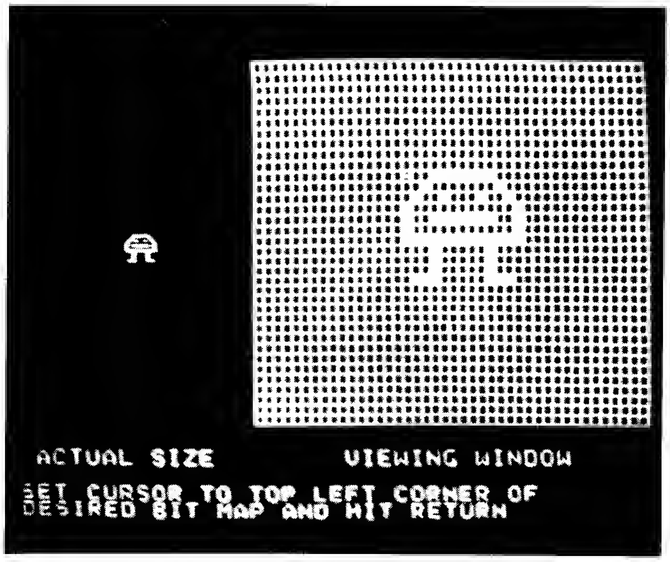


Photo 3: *A view of the first step in forming a shape table. The desired shape is selected by defining a rectangle enclosing the shape. Here, the user has positioned the cursor to the correct position to define the upper left corner of the rectangle.*

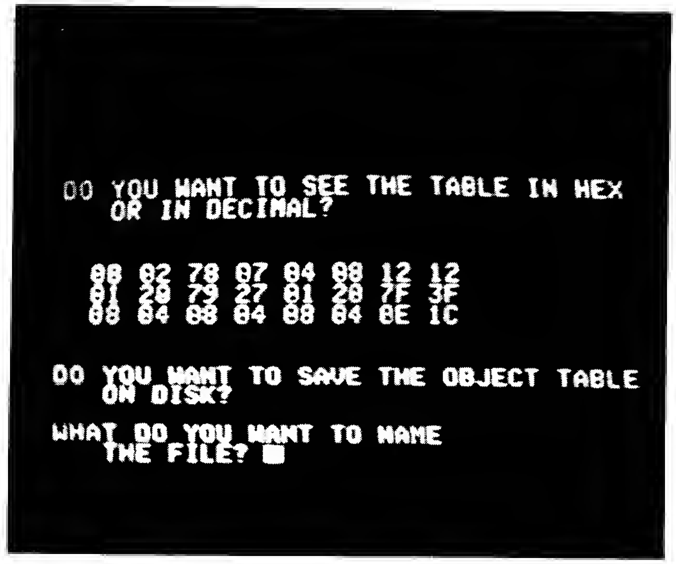


Photo 4: *A view of the screen after the shape-editor program has formed the shape table for the shape shown in photo 3.*

pending on how you answer a prompt. The program will then save this object-file shape table on disk as a standard binary file if you so desire. You are then asked whether to return to the screen-edit mode or end the program. Photo 4 shows the final shape table formed from the sample shape used in photo 3.

The Save and Get commands let you store on disk and later retrieve any picture drawn in the screen-edit mode. The Save command prompts you for a file name and then saves to disk a representation of the shape drawn on the grid. The Get command can then be used to retrieve and display the picture as long as the saved file remains on disk. Because the Get command erases any drawing previously on the screen, you are first asked to confirm that a file is to be loaded. Once the picture is retrieved, it can be modified or assembled into a shape table just as if the picture had been entered using the keyboard commands.

The Help command (executed by pressing the H or ? key) returns you from the screen-edit mode to the menu shown at the beginning of the program for a quick command-letter check. Pressing the space bar returns you to screen-edit mode with the contents of the screen unaltered. The Quit command ends the program. Because any drawing on the screen is lost once the program is ended, you are asked to confirm the Quit directive.

Summing Up

Using the techniques and programs described in this article, you can implement professional-looking animation on the Apple without having to work around the limitations of the standard Apple shape subroutine. Although I wrote my shape subroutine with animation in mind, the subroutine is useful in any graphics applications where detailed shapes must be drawn. Using the graphics editor as a development tool, virtually any shape can be easily displayed on the hi-res screen.■

Richard T. Simoni Jr. (29 Farnham Park Dr., Houston, TX 77024) is currently enrolled as a senior electrical engineering/computer science/math science major at Rice University in Houston, Texas.
